# Nearly Maximum Flows in Nearly Linear Time

Jonah Sherman*
University of California, Berkeley

April 9, 2013(preliminary draft)

**Abstract**

We introduce a new approach to the maximum flow problem in undirected, capacitated graphs using $\alpha$-*congestion-approximators*: easy-to-compute functions that approximate the congestion required to route single-commodity demands in a graph to within a factor of $\alpha$. Our algorithm maintains an arbitrary flow that may have some residual excess and deficits, while taking steps to minimize a potential function measuring the congestion of the current flow plus an over-estimate of the congestion required to route the residual demand. Since the residual term over-estimates, the descent process gradually moves the contribution to our potential function from the residual term to the congestion term, eventually achieving a flow routing the desired demands with nearly minimal congestion after $\tilde{O}(\alpha\varepsilon^{-2}\log^2 n)$ iterations. Our approach is similar in spirit to that used by Spielman and Teng (STOC 2004) for solving Laplacian systems, and we summarize our approach as trying to do for $\ell_\infty$-flows what they do for $\ell_2$-flows.

Together with a nearly linear time construction of a $n^{o(1)}$-congestion-approximator, we obtain $1+\varepsilon$-optimal single-commodity flows undirected graphs in time $m^{1+o(1)}\varepsilon^{-2}$, yielding the fastest known algorithm for that problem. Our requirements of a congestion-approximator are quite low, suggesting even faster and simpler algorithms for certain classes of graphs. For example, an $\alpha$-competitive oblivious routing tree meets our definition, *even without knowing how to route the tree back in the graph.* For graphs of conductance $\phi$, a trivial $\phi^{-1}$-congestion-approximator gives an extremely simple algorithm for finding $1+\varepsilon$-optimal-flows in time $\tilde{O}(m\phi^{-1})$.

## 1 Introduction

The maximum flow problem and its dual, the minimum cut problem are fundamental combinatorial optimization problems with a wide variety of applications. In the well-known maximum $s-t$ flow problem we are given a graph $G$ with edge capacities $c_e$, and aim to route as much flow as possible from $s$ to $t$ while restricting the magnitude of the flow on each edge to its capacity. We will prefer instead to think in terms of the equivalent problem of routing a single unit of flow from $s$ to $t$ while minimizing the maximum congestion $|f_e/c_e|$ on any edge; clearly the minimum congestion for unit flow is equal to one divided by the maximum flow of congestion one. Once formulated that way, we need no longer restrict ourselves to $s-t$ flows; given a *demand vector* $b \in \mathbb{R}^n$ specifying the excess desired at each vertex, we aim to find a flow $f \in \mathbb{R}^m$ with divergence equal to $b$ that minimizes the maximum congestion $|f_e/c_e|$.[1]

In this paper, we introduce a new approach to this problem in undirected graphs. We maintain a flow that may not quite route $b$ exactly, but we also keep track of an upper bound on how much it will cost us in congestion to fix it back up. We will aim to minimize a potential function measuring the current congestion plus an over-estimate on the cost of fixing up the residuals. By not needing to worry about precisely conserving flow at every vertex, we can take large steps in each iteration towards minimizing our potential function. On the other hand, by intentionally over-estimating the cost of fixing up the residuals, in the course of minimizing our potential function we must inevitably fix them up, as it will cost strictly less to do so.

For a graph $G$, let $C$ be the $m \times m$ diagonal matrix containing the edge capacities, and let $B$ be the $n \times m$ divergence matrix, where $(Bf)_i$ is the excess at vertex $i$. For a set $S \subseteq V$, we'll write $b_S = \sum_{i \in S} b_S$, the total excess in $S$, and $c_S = \sum_{e:S \leftrightarrow V\setminus S} c_e$, the capacity of the cut $(S, V \setminus S)$ in $G$. A valid demand vector satisfies $b_V = 0$.

*Research supported in part by a Microsoft Graduate Fellowship and in part by a NSF Graduate Fellowship

[1]In fact the $s-t$ case is no less general, since one could always add a new vertices $s$ and $t$, connect each $v$ to $s$ or $t$ according to the sign of $b_v$ with an edge of capacity $\beta|b_v|$ and scale $\beta$ until the additional edges are saturated.

The minimum congestion flow problem for demands $b$, and its dual, the maximum congested cut, are

$$\min \quad \|C^{-1}f\|_\infty \quad s.t.\ Bf = b \tag{1}$$
$$\max \quad b^\top v \quad s.t.\ \|CB^\top v\|_1 \leq 1 \tag{2}$$

We refer to the optimum value of these problems as $\mathsf{opt}(b)$. It is well-known that for problem (2), one of the threshold cuts with respect to $v$ achieves $b_S/c_S \geq b^\top v$.

## 1.1 History

Much of the early work on this problem considers the general, directed edge case, culminating in the still-best binary blocking flow algorithm of Goldberg and Rao[4] that achieves $\tilde{O}(m \min(m^{1/2}, n^{2/3})$ time. Karger and Levine[6] give evidence that the undirected case seems easier in graphs with small flow values. The smooth sparsification technique of Benczúr and Karger[1] shows one can split a graph with $m$ edges into $t = \tilde{O}(m\varepsilon^2/n)$ graphs each with with $\tilde{O}(n\varepsilon^{-2})$ edges, and each of which has at least $(1-\varepsilon)/t$ of the capacity of the original graph. Therefore, for undirected graphs, any algorithm running in time $T(m,n)$ can be replaced with one running in time $\tilde{O}(m) + (m/n)T(\tilde{O}n, n)$. Using the algorithm of Goldberg and Rao yields approximate maximum flows in $\tilde{O}(mn^{1/2})$ time, and for many years that was the best known.

In a breakthrough, Christiano *et al.* show how to compute approximate maximum flows in $\tilde{O}(mn^{1/3})$ time[3]. Their new approach uses the nearly linear-time Laplacian solver of Spielman and Teng[12] to take steps in the minimization of a softmax approximation of the edge congestions. Each step involves minimizing $\|WC^{-1}f\|_2$, a weighted $\ell_2$-norm of the congestions. While a naive analysis yields immediately yields a method that makes $\tilde{O}(\sqrt{m})$ such iterations (because $\|\cdot\|_2$ approximates $\|\cdot\|_\infty$ by a $\sqrt{m}$ factor), they present a surprising and insightful analysis showing in fact only $\tilde{O}(m^{1/3})$ such $\ell_2$ iterations suffice. The maximum-flow-specific parts of [3] are quite simple, needing only to maintain the weights $W$ and then using the Spielman-Teng solver as a black box.

## 1.2 Outline

In this work we pry open that box, extract the parts we need, and apply them *directly* to the maximum flow problem. In the course of doing so, we push the running time for this decades-old problem nearly down to linear. Solving a Laplacian system $Lv^* = b^*$, where $L = BCB^\top$, corresponds to finding a flow $f^*$ with $Bf^* = b^*$ minimizing $\|C^{-1/2}f\|_2$. While Spielman and Teng work entirely in the dual space of vertex potentials and never explicitly represent flows, we can still translate between spaces throughout the algorithm to get an idea of what is actually going on. At any point, the vertex potentials $v$ induce an optimal flow for *some* demands $b = Lv$; just perhaps not the ones desired. The solution of Spielman and Teng is to maintain a simpler graph $G'$ that approximates $G$, in the sense that the $\ell_2$ cost of routing flows in $G'$ is within some factor $\alpha$ of the cost in $G$. The residual flow is routed optimally in $G'$(recursively), by solving $L'v' = b^* - b$. The potentials that induced that flow are added to $v$, in the hope that $Lv + Lv' \approxeq Lv + L'v' = b^*$. Indeed, if $G'$ approximates $G$ by a factor $\alpha$, then $b^* - b$ gets smaller in a certain norm by $(1 - 1/\alpha)$, nudging the flow towards actually routing $b^*$.

Our first step towards obtaining a ST-like algorithm is the definition of a good approximator for the congestion required by $\ell_\infty$-flows.

**Definition 1.1.** *An $\alpha$-congestion-approximator for $G$ is a matrix $R$ such that for any demand vector $b$,*

$$\|Rb\|_\infty \leq \mathsf{opt}(b) \leq \alpha\|Rb\|_\infty$$

Our main result is that we can use good congestion approximators to quickly find near-optimal flows in a graph. We prove the following theorem in section 2.

**Theorem 1.2.** *There is an algorithm that, given demands $b$ and access to an $\alpha$-congestion-approximator $R$, makes $\tilde{O}(\alpha\varepsilon^{-2}\log^2(n))$ iterations and then returns a flow $f$ and cut $S$ with $Bf = b$ and $\|C^{-1}f\| \leq (1+\varepsilon)b_S/c_S$. Each iteration requires $O(m)$ time, plus a multiplication by $R$ and $R^\top$.*

For the sake of giving the reader a concrete example of what a congestion-approximator might look like before continuing, we'll begin with two simple toy examples.

**Example 1.3.** *Let $T$ be a maximum weight spanning tree in $G$, and let $R$ be the $n-1 \times n$ matrix with a row for each edge in $T$, and*

$$(Rb)_e = \frac{b_S}{c_S}$$

*where $(S, V \setminus S)$ is the cut in $G$ induced by removing $e$ from $T$.*

*Then, $R$ is a $m$-congestion-approximator.*

*Proof.* Since $(Rb)_e$ is the congestion on the cut in $G$ induced by removing $e$ from $T$, certainly $\mathsf{opt}(b) \geq \|Rb\|_\infty$. On the other hand, at least $1/m$ of the capacity of those cuts is contained in $T$, so routing $b$ through $T$ congests $e$ by at most $m|(Rb)_e|$. Multiplication by $R$ and $R^\top$ can be done in $O(n)$ time via elimination on leaves. □

For graphs of large conductance, we can obtain a trivial approximator by simply looking at how much the demand into each vertex congests its total degree.

**Example 1.4.** *Let $G$ have conductance $\phi$. Let $R$ be $n \times n$ diagonal matrix with $R_{i,i} = 1/\deg(i)$ where $\deg(i) = c_{\{i\}}$. Then, $R$ is a $\phi^{-1}$-congestion-approximator.*

*Proof.* Routing $|b_i|$ into or out of vertex $i$ certainly must congest one of its edges by at least $|b_i|/\deg(i)$, so $\mathsf{opt}(b) \geq \|Rb\|_\infty$. On the other hand, the capacity of any cut in $G$ is at least $\phi$ times the total degree of the smaller side. It follows that if no vertex is congested by more than $\beta$, then no cut is congested by more than $\phi^{-1}\beta$. □

Those two simple examples are analogous to the simple cases for $\ell_2$ flows in the ST-algorithm. The former is analogous to preconditioning by a small-stretch spanning tree, while the latter is analogous to not preconditioning at all. As in the ST-algorithm, those two simple examples in fact capture the ideas behind our real constructions. In section 3, we show how to apply the *$j$-tree decomposition* of Madry[10] (itself based on the ultrasparsifiers of Spielman and Teng[12])) to obtain good congestion approximators for any graph.

**Theorem 1.5.** *For any $1 \leq k \leq \log n$, there is an algorithm to construct a data structure representing a $\log(n)^{O(k)}$-congestion-approximator $R$ in time $\tilde{O}(m + n^{1+1/k})$. Once constructed, $R$ and $R^\top$ can each be applied in time $\tilde{O}(n^{1+1/k})$.*

Combining theorem 1.5 where $k = \tilde{\theta}(\sqrt{\log n})$ with theorem 1.2 yields our main result of a nearly-linear time algorithm for minimum congestion flows.

**Theorem 1.6.** *There is an algorithm to compute $(1+\varepsilon)$-approximate minimum congestion flows in time $m\varepsilon^{-2} \cdot \exp(\tilde{O}(\sqrt{\log n}))$. For graphs of conductance $\phi$, the same can be done in time $\tilde{O}(m\phi^{-1})$.*

## 2 Congestion Potential

The key to our scheme lies transforming problem (1) to an unconstrained optimization problem, using the congestion-approximator to bound the cost of routing the residual. To that end, we introduce our potential function.

$$\min \|C^{-1}f\|_\infty + 2\alpha\|R(b - Bf)\|_\infty \tag{3}$$

We believe the *mere statement* of the potential function (3) to be the most important idea in this paper. Once (3) has been written down, the remainder of our algorithm is nearly obvious. We solve problem (3) nearly-optimally by approximating $\|\cdot\|_\infty$ with a softmax. The softmax is well-approximated by its gradient in the region where steps are taken with $\ell_\infty$-norm $O(1)$. Since $\|RBC\|_{\infty\to\infty} \leq 1$ for a congestion-approximator, we can take steps on edge $e$ of size $\Omega(\alpha^{-1})c_e$. We include the details at the end of this section, proving the following theorem.

**Theorem 2.1.** *There is an algorithm `AlmostRoute` that given $b$ and $\varepsilon \leq 1/2$, performs $\tilde{O}(\alpha\varepsilon^{-2}\log n)$ iterations and returns a flow $f$ and cut $S$ with,*

$$\|C^{-1}f\|_\infty + 2\alpha\|R(b - Bf)\|_\infty \leq (1+\varepsilon)\frac{b_S}{c_S}$$

*Each iteration requires $O(m)$ time plus a multiplication by $R$ and $R^\top$.*

The flow $f$ may not quite route the demands we wanted. Fortunately, that will be easy to fix. The extra factor of two in equation (3) means that half of the contribution to the objective value from the residual part is pure slack. On the other hand, if $f$ is nearly optimal, there can't be too much slack.

**Lemma 2.2.** *Suppose* $\|C^{-1}f\|_\infty + 2\alpha\|R(b - Bf)\|_\infty \leq (1+\varepsilon)\mathtt{opt}(b)$. *Then,* $\|R(b - Bf)\|_\infty \leq \varepsilon\|Rb\|_\infty$.

*Proof.* Let $f$ meet the assumption. Let $f'$ be a routing of $b - Bf$ in $G$ with $\|C^{-1}f'\|_\infty \leq \alpha\|R(b - Bf)\|_\infty$. Then, moving from $f$ to $f + f'$ decreases the objective value by atleast $\alpha\|R(b - Bf)\|$. On the other hand, that decrease can't exceed $\varepsilon\mathtt{opt}(b)$. Since $\mathtt{opt}(b) \leq \alpha\|Rb\|_\infty$, the lemma follows. □

So while `AlmostRoute` may not route $b$, our bound for the congestion to route the residual is at most half of our bound to route the original demands. Furthermore, the objective already pays the cost of routing that residual. In fact, it pays it with a factor of two, so we need only route the remaining residual within a factor-two of optimal. That suggests an obvious way to route demands $b$: repeatedly invoke `AlmostRoute` on the remaining residual, until the congestion required to route it is extremely small compared to the congestion required to route $b$. Then route the final residual in a naive way, such as via a maximal spanning tree. The cost of that final routing will be paid for by the slack in the objective value of the first routing, simply by finding a factor 3/2-optimal routing for each residual after the first case.

Formalizing the latter argument completes our proof of theorem 1.2. Set $b_0 \leftarrow b$, and let $(f_0, S_0) \leftarrow$ `AlmostRoute`$(b_0, \varepsilon)$. Next for $i = 1, \ldots, T$ where $T = \log(2m)$, set $b_i \leftarrow b_{i-1} - Bf_{i-1}$ and $(f_i, S_i) \leftarrow$ `AlmostRoute`$(b_i, 1/2)$ (we don't actually need any $S_i$ after $S_0$). Finally, let $b_{T+1} = b_t - Bf_t$, and let $f_{T+1}$ be a flow routing $b_{T+1}$ in a maximal spanning tree of $G$. Output $f_1 + \cdots + f_{T+1}$ and $S_0$. Observe that theorem 2.1 yields

$$\begin{aligned}\|C^{-1}f_0\|_\infty + 2\alpha\|Rb_1\|_\infty &\leq (1+\varepsilon)b_{S_0}/c_{S_0} \\ \|C^{-1}f_i\|_\infty + 2\alpha\|Rb_{i+1}\|_\infty &\leq (3/2)\mathtt{opt}(b_i) \leq (3/2)\alpha\|Rb_i\|_\infty\end{aligned}$$

Beginning with the former inequality and repeatedly applying the latter yields,

$$\begin{aligned}(1+\varepsilon)\frac{b_{S_0}}{c_{S_0}} &\geq \|C^{-1}f_0\|_\infty + 2\alpha\|Rb_1\|_\infty \\ &\geq (1/2)\alpha\|Rb_1\|_\infty + \|C^{-1}f_0\|_\infty + \|C^{-1}f_1\|_\infty + 2\alpha\|Rb_2\|_\infty \\ &\vdots \\ &\geq (1/2)\alpha\|Rb_1\|_\infty + \|C^{-1}f_0\|_\infty + \cdots + \|C^{-1}f_T\|_\infty + 2\alpha\|Rb_{T+1}\|_\infty\end{aligned}$$

On the other hand, by choice of $T$, we have

$$\|C^{-1}f_{T+1}\|_\infty \leq m\alpha\|Rb_{T+1}\|_\infty \leq m\alpha 2^{-T}\|Rb_1\|_\infty \leq (1/2)\alpha\|Rb_1\|_\infty$$

Combining the two yields the theorem.

## 2.1 Proof of Theorem 2.1

For this preliminary draft, we prove theorem 2.1 with slightly worse parameters of $\tilde{O}(\alpha^2\varepsilon^{-3}\log^2(n))$, using naive steepest descent. The better parameters follow from using Nesterov's accelerated gradient method[11] instead, as will be proved in the final version of this paper. Of course, for the case of $\alpha = \exp(\tilde{O}(\sqrt{\log n}))$, then still $\alpha^2 = \exp(\tilde{O}(\sqrt{\log n}))$, so this naive analysis still yields nearly linear-time flow algorithms.

We approximate $\|\cdot\|_\infty$ using the symmetric softmax function.

$$\mathrm{lmax}(x) = \log\left(\sum_i e^{x_i} + e^{-x_i}\right)$$

We make use of some elementary facts about lmax.

**Fact 2.3.** *Let* $x, y \in \mathbb{R}^d$. *Then,*

$$\begin{aligned}\|\nabla\,\mathrm{lmax}(x)\|_1 &\leq 1 &(4)\\ \nabla\,\mathrm{lmax}(x)^\top x &\geq \mathrm{lmax}(x) - \log(2d) &(5)\\ \|\nabla\,\mathrm{lmax}(x) - \nabla\,\mathrm{lmax}(y)\|_1 &\leq \|x - y\|_\infty &(6)\end{aligned}$$

We will approximate problem (3) with the potential function,

$$\phi(f) = \text{lmax}(C^{-1}f) + \text{lmax}\left(2\alpha R(b - Bf)\right) \tag{7}$$

Since equation (7) approximates equation (3) to within an additive $\theta(\log n)$, we will be concerned with minimizing $\phi(f)$ after scaling $f, b$ so $\phi(f) = \theta(\varepsilon^{-1}\log n)$.

```
AlmostRoute(b, ε):
• Initialize $f = 0$, scale $b$ so $2\alpha\|Rb\|_\infty = 16\varepsilon^{-1}\log(n)$.
• Repeat:
    – While $\phi(f) < 16\varepsilon^{-1}\log(n)$, scale $f$ and $b$ up by 17/16.
    – Set $\delta \leftarrow \|C\nabla\phi(f)\|_1$.
    – If $\delta \geq \varepsilon/4$, set $f_e \leftarrow f_e - \frac{\delta}{1+4\alpha^2}\,\text{sgn}(\nabla\phi(f)_e)c_e$
    – Otherwise, terminate and output $f$ together with the potentials induced by $\nabla\phi(f)$(see below),
      after undoing any scaling.
```

Each step requires computing $\nabla\phi(f)$, which requires $O(m)$ time plus a multiplication by $R$ and a multiplication by $R^\top$. Further, the partial derivative of the residual part for a particular edge is equal to a potential difference between the endpoints of that edge. When $\phi(f)$ is nearly-optimal, those potentials yield a good dual solution for our original problem.

**Lemma 2.4.** *When* `AlmostRoute` *terminates, we have a flow $f$ and potentials $v$ with,*

$$\|C^{-1}f\|_\infty + 2\alpha\|R(b - Bf)\|_\infty \leq (1+\varepsilon)\frac{b^\top v}{\|CB^\top v\|_1}$$

*Proof.* Set $x_1 = C^{-1}f$, $x_2 = 2\alpha R(b-Bf)$, and $p_i = \nabla\,\text{lmax}(x_i)$. Set $v = R^\top p_2$ to be our potentials. Observe that $\nabla\phi(f) = C^{-1}p_1 - 2\alpha B^\top v$. First, equation (4) yields

$$2\alpha\|CB^\top v\|_1 \leq \|p_1\|_1 + \|p_1 - 2\alpha CB^\top v\|_1 \leq 1 + \delta$$

By equation (5), using the fact that $C$ and $R$ have at most $n^2/2$ rows and $\phi(f) \geq 16\varepsilon^{-1}\log n$,

$$p_1^\top C^{-1}f + 2\alpha p_2^\top R(b - Bf) \geq \phi(f) - 4\log n \geq \phi(f)(1 - \varepsilon/4)$$

On the other hand,

$$\begin{aligned}
\delta\phi(f) &\geq \|C\nabla\phi(f)\|_1\|C^{-1}f\|_\infty \\
&\geq \nabla\phi(f)^\top f \\
&= (C^{-1}p_1 - 2\alpha B^T R^T p_2)^\top f \\
&= p_1^\top C^{-1}f - 2\alpha p_2^\top RBf
\end{aligned}$$

Combining the two yields,

$$2\alpha b^\top v \geq \phi(f)(1 - \varepsilon/4 - \delta)$$

Altogether, using the fact that $\delta < \varepsilon/4$ at termination, we have

$$\frac{b^\top v}{\|CB^\top v\|_1} \geq \frac{\phi(f)(1-\varepsilon/2)}{1+\varepsilon/4} \geq \frac{\phi(f)}{1+\varepsilon}$$

Observing that $\phi(f)$ overestimates $\|C^{-1}f\|_\infty + 2\alpha\|R(b - Bf)\|_\infty$ completes the proof. □

**Lemma 2.5.** `AlmostRoute` *terminates after at most $\tilde{O}(\alpha^2\varepsilon^{-3}\log n)$ iterations.*

*Proof.* Let us call the iterations between each scaling a *phase.* Since $\|Rb\|_\infty$ gives us the correct scale to within factor $\alpha$, we will scale at most $O(\log\alpha)$ times.

Let $h_e = -\frac{\delta}{1+4\alpha^2}\operatorname{sgn}(\nabla_f \phi(f)_e)c_e$ be our step. Then, equation (6), together with the fact that $\|RBC\|_{\infty\to\infty} \leq 1$ for a congestion-approximator $R$ yields,

$$\begin{aligned}\phi(f+h) &\leq \phi(f) + \nabla\phi(f)^\top h + \frac{1+4\alpha^2}{2}\|C^{-1}h\|_\infty^2 \\ &= \phi(f) - \frac{\delta^2}{2+8\alpha^2} \\ &= \phi(f) - \Omega(\varepsilon^2\alpha^{-2})\end{aligned}$$

Since we raised $\phi(f)$ by at most $\varepsilon^{-1}\log n$ when scaling, and each step drops $\phi(f)$ by at least $\Omega(\varepsilon^2\alpha^{-2})$, there can be at most $O(\alpha^2\varepsilon^{-3}\log n)$ steps between phases. □

## 3 Computing Congestion-Approximators

In this section we prove theorem 1.5, using a construction of Madry[10], itself based on a construction of Spielman and Teng[12].

**Definition 3.1** (Madry[10])**.** *A $j$-tree is a graph formed by the union of a forest with $j$ components, together with a graph $H$ on $j$ vertices, one from each component. The graph $H$ is called the core.*

**Theorem 3.2** (Madry[10])**.** *For any graph $G$ and $t \geq 1$, we can find in time $\tilde{O}(tm)$ a distribution of $t$ graphs $(\lambda_i, G_i)$ such that,*

- *Each $G_i$ is a $O(m \log m/t)$-tree, with a core containing at most $m$ edges.*
- *$G_i$ dominates $G$ on all cuts.*
- *$\sum_i \lambda_i G_i$ can be routed in $G$ with congestion $\tilde{O}(\log n)$.*

We briefly remark that while the statement of theorem 3.2 in [10] contains an additional logarithmic dependence on the *capacity-ratio* of $G$, that dependence is easily eliminated. We elaborate further in appendix A. Our construction will simply apply theorem 3.2 recursively, sparsifying the core on each iteration. To accomplish that, we use an algorithm of Benczúr and Karger[1].

**Theorem 3.3** (Benczúr, Karger[1])**.** *There is an algorithm* `Sparsify`$(G, \varepsilon)$ *that, given a graph $G$ with $m$ edges, takes $\tilde{O}(m)$ time and returns a graph $G'$ with $m' = O(n\varepsilon^{-2}\log n)$ edges such that the capacity of cuts in the respective graphs satisfy*

$$G \leq G' \leq (1+\varepsilon)G$$

*Further, the edges of $G'$ are scaled versions of a subset of edges in $G$, with no edge scaled by more than $(1+\varepsilon)m/m'$.*

We now present the algorithm for computing the data structure representing a congestion-approximator. The algorithm `ComputeTrees` assumes its input is sparse; our top-level data-structure is constructed by invoking `ComputeTrees`(`Sparsify`$(G, 1), n^{1/k}$), where $k$ is the parameter of theorem 1.5.

```
ComputeTrees($G, t$):
  • If $n = 1$, return.
  • Using theorem 3.2, compute distribution $(\lambda_i, G_i)_i^{t'}$ of $\max(1, n/t)$-trees.
  • Pick the $t$ graphs of largest $\lambda_i$, throw away the rest, and scale the kept $\lambda_i$ to sum to 1.
  • For $i = 1, \ldots, t$:
      – $H'_i \leftarrow$ Sparsify($H_i, 1$), where $H_i$ is the core of $G_i$
      – $L_i \leftarrow$ ComputeTrees($H'_i, t$)
  • Return the list $L = (\lambda_i, F_i, L_i)_{i=1}^t$ where $F_i$ is the forest of $G_i$.
```

The analysis of `ComputeTrees` correctness will make use of another algorithm for sampling trees. The `SampleTree` procedure is only used for analysis, and is not part of our flow algorithm.

```
SampleTree($L = (\lambda_i, F_i, T_i)_i^t$):
    • Pick $i$ with probability $\lambda_i$.
    • Output $F_i$ + SampleTree($T_i$)
```

**Lemma 3.4.** *Let $G$ have $\tilde{O}(n)$ edges, and set $L \leftarrow$ `ComputeTrees`$(G, t)$. Then, every tree in the sample space of* `SampleTree`$(L)$ *dominates $G$ on all cuts, and* $\mathbf{E}[$`SampleTree`$(L)]$ *is routable in $G$ with congestion $\log(n)^{\log(n)/log(t)}$. Further, the computation of $L$ takes $\tilde{O}(tn)$ time.*

*Proof.* By induction on $n$. For $n = 1$ the claim is vacuous, so suppose $n = t^{k+1}$. Since $G$ has $O(n \log n)$ edges, the distribution output by theorem 3.2 will have $O(t \log^2 n)$ entries. We have $H_i' \geq H_i$ and $H_i + F_i \geq G$. Furthermore, the inductive hypothesis implies that every tree $T_i$ in `SampleTree`$(L_i)$ dominates $H_i'$. Then,

$$T_i + F_i \geq H_i' + F_i \geq H_i + F_i = G_i \geq G$$

Sparsifying the distribution from $O(t \log^2 n)$ to $t$ scales $\lambda_i$ by at most $O(\log^2 n)$, so that $\sum_{i=1}^t \lambda_i G_i$ is routable in $G$ with congestion at most $\log^2 n$ larger than the original distribution. Since $H_i' \leq 2H_i$, by the multicommodity max-flow/min-cut theorem[9] $H_i'$ is routable in $H_i$ with congestion $O(\log n)$. By the inductive hypothesis, $\mathbf{E}[$`SampleTree`$(L_i)]$ is routable in $H_i'$ with congestion $\log^{O(k)}(n)$. It follows then that $\mathbf{E}[$`SampleTree`$(L)]$ is routable in $G$ with congestion at most $\log^{O(k+1)}(n)$.

Finally, `ComputeTrees` requires $\tilde{O}(tn)$ time to compute the distribution, another $\tilde{O}(tn)$ time to sparsify the cores, and then makes $t$ recursive calls on sparse graphs with $n/t$ vertices. It follows that the running time of `ComputeTrees` is $\tilde{O}(tn)$. □

**Lemma 3.5.** *Let $R$ be the matrix that has a row for each forest edge in our data structure, and $(Rb)_e$ is the congestion on that edge when routing $b$. If* $\mathbf{E}[$`SampleTree`$(L)]$ *is routable in $G$ with congestion $\alpha$, then $R$ is a $\alpha$-congestion-approximator for $G$. Further, $R$ has $\tilde{O}(tn)$ rows.*

*Proof.* Since the capacity of each tree-edge dominates the capacity of the corresponding cut in $G$, $\mathsf{opt}(b) \geq \|Rb\|_\infty$. On the other hand, $b$ can be routed in every tree with congestion $\|Rb\|_\infty$. By routing a $\mathbf{Pr}[T]$ fraction of the flow through tree $T$, we route $b$ in $\mathbf{E}[$`SampleTree`$(L)]$ with congestion $\|Rb\|_\infty$. But then $b$ can be routed in $G$ while congesting by at most an $\alpha$ factor larger.

The total number of edges in $R$ satisfies the recurrence $E(n) \leq nt + tE(n/t)$ as each edge is either in one of the $t$ toplevel forests, or in one of the $t$ subgraphs. □

Having constructed our representation of $R$, it remains only to show how to multiply by $R$ and $R^\top$. We use the following lemmas as subroutines, which are simple applications of leaf-elimination on trees.

**Lemma 3.6.** *There is an algorithm* `TreeFlow` *that, given a tree $T$ and a demand vector $b$, takes $O(n)$ time and outputs for each tree edge, the flow along that edge when routing $b$ in $T$.*

**Lemma 3.7.** *There is an algorithm* `TreePotential` *that, given a tree $T$ annotated with a price $p_e$ for each edge, takes $O(n)$ time and outputs a vector of vertex potentials $v$ such that, for any $i, j$, the sum of the prices on the path from $j$ to $i$ in $T$ is $v_i - v_j$.*

We begin with computing $R$. We take as input the demand vector $b$, and then annotate each forest edge $e$ with the congestion $r_e$ induced by routing $b$ through a tree containing $e$.

```
Compute$R(b, L = (\lambda_i, F_i, T_i)_i^t)$:
  • For $i = 1, \ldots, t$:
      – Let $T$ be the tree formed by taking $F_i$, adding a new vertex $s$, and an edge from $s$ to each
        core-vertex of $F_i$. Augment $b$ with demand zero to the new vertex.
      – $f \leftarrow$ TreeFlow$(b, T)$.
      – Set $r_e \leftarrow f_e/c_e$ for each forest edge in $F_i$.
      – Set $b'$ to a vector indexed by core-vertices, with $b'_j$ equal to the flow on the edge from $s$ to
        core-vertex $j$.
      – ComputeR$(b', T_i)$.
```

**Lemma 3.8.** *The procedure* Compute$R(b, L)$ *correctly annotates each edge $e$ with $r_e = (Rb)_e$, and takes $\tilde{O}(tn)$ time.*

*Proof.* Let $L = (\lambda_i, F_i, T_i)_i^t$. We argue by induction on the depth of recursion. Fix a level and index $i$. Observe that the cut in $G$ induced by cutting a forest edge is the same regardless of what tree $T$ lies on the core: it is the cut that separates the part of $F_i$ not containing the core from the rest of the vertices. It follows that we may place *any* tree on the core vertices, invoke TreeFlow, and obtain the flow on each forest edge. Next, for each component $S$ of $F_i$, the total excess $b_S$ must enter $S$ via the core vertex. It follows that in a flow routing $b$ on $F_i + T'$, for any tree $T'$, the restriction of that flow to $T'$ must have excess $b_S$ on the core vertex of $S$, so it suffices to find a flow in the core with demands $b'_j = b_{S_j}$. But routing $b$ in $F_i + T$ will place exactly $b_{S_j}$ units of flow on the edge from $s$ to core-vertex $j$.

The running time consists of $t$ invocations of TreeFlow each taking $O(n)$ time, plus $t$ recursive calls on graphs of size $n/t$, for a total running time of $\tilde{O}(tn)$. □

To compute $R^\top$, we assume each forest edge $e$ has been annotated with a price $p_e$ that must be paid by any flow per unit of congestion on that edge, and output potentials $v$ such that $v_i - v_j$ is the total price to be paid for routing a unit of flow from $j$ to $i$.

```
Compute$R^\top(L = (\lambda_i, F_i, T_i)_i^t)$:
  • $v \leftarrow 0$
  • For $i = 1, \ldots, t$:
      – $v' \leftarrow$ Compute$R^\top(T_i)$.
      – Let $T$ be the tree formed by taking $F_i$, adding a vertex $s$, and an edge from $s$ to each core-vertex
        of $F_i$. Set $q_e = p_e/c_e$ for each forest edge, and $q_e = v'_j$ for edge $e$ from $s$ to core-vertex $j$.
      – $v'' \leftarrow$ TreePotential$(T, q)$
      – Add $v''$ to $v$ after removing the entry for $s$.
  • Return $v$
```

**Lemma 3.9.** *Given edges annotated with per-congestion prices, the procedure* Compute$R^\top(L)$ *correctly returns potentials $v$ such that $v_k - v_j$ is the cost per unit of flow from vertex $j$ to $k$.*

*Proof.* Let $L = (\lambda_i, F_i, T_i)_i^t$. We argue by induction on the depth of recursion. Fix a level; a flow must pay its toll to each $G_i$, so the resulting potential equals the sum of the potentials for each $i$. Fix an index $i$. A unit of flow from $j$ to $k$ is first routed from $j$ to the core-vertex of the component of $F_i$ containing $j$, then to the core-vertex of the component containing $k$, and then finally to $k$. By induction, we assume that $v'$ yields potentials that give the per-unit costs of routing between core-vertices. Placing a star on the core with the edge from $s$ to core-vertex $j$ having per-unit cost $v'_j$ preserves those costs. If $p_e$ is the price of an edge per unit of congestion, then $q_e = p_e/c_e$ is the price of an edge per unit of flow. It follows that the total toll paid is the same as the toll paid in $T$; thus, the potentials output by TreePotential$(T, q)$ are correct.

The running time consists of $t$ recursive calls to Compute$R^\top$ on graphs of size $n/t$, plus $t$ invocations of TreePotential each taking $O(n)$ time, for a total running time of $\tilde{O}(tn)$. □

## 4 Final Remarks

We remark that there are many other ways to obtain good congestion approximators. The oblivious routing schemes of [5, 2] require polynomial time to compute, but, once computed, give us a single tree whose single-edge cuts yield a $\log(n)^{O(1)}$-congestion approximator. Furthermore, we only need the actual tree, and not the routings of the tree back in the original graph. If such a single tree could be computed in nearly-linear time, it would make an ideal candidate for use in our algorithm.

There have been substantial simplifications to Spielman and Teng's original algorithm (see [8, 7]). It may be possible to use some of those techniques to further simplify our algorithm.

## References

[1] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *CoRR*, cs.DS/0207078, 2002.

[2] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *SPAA*, pages 24–33. ACM, 2003.

[3] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 273–282. ACM, 2011.

[4] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[5] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '03, pages 34–43, New York, NY, USA, 2003. ACM.

[6] David R. Karger and Matthew S. Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In Jeffrey Scott Vitter, editor, *STOC*, pages 69–78. ACM, 1998.

[7] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. *CoRR*, abs/1301.6628, 2013.

[8] Ioannis Koutis, Gary L. Miller, and Richard Peng. A fast solver for a class of linear systems. *Commun. ACM*, 55(10):99–107, 2012.

[9] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS*, pages 245–254. IEEE Computer Society, 2010.

[11] Yu Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, May 2005.

[12] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

## A Fixing Theorem 3.2

The proof of theorem 3.2 maintains a length function $l(e)$ for each edge, and repeatedly invokes an algorithm `SmallStretchTree` that returns a spanning tree $T$ on $G$ with,

$$\sum_e l_T(e)c(e) \leq \tilde{O}(\log n) \sum_e l(e)c(e) \tag{8}$$

where $l_T(e)$ is the length of the path between $e$'s endpoints in the tree. Without loss of generality, by scaling, we assume $\sum_e l(e)c(e) = m$. Let $\chi(e, e') = 1$ if $e'$ is a tree edge that lies on the path in $T$ containing $e$. Then,

$$\sum_e l_T(e)c(e) = \sum_e c(e) \sum_{e'} chi(e, e')l(e') = \sum_{e'} l(e') \sum_e \chi(e, e')c(e) = \sum_{e'} l(e')c_T(e')$$

where $c_T(e')$ is the total capacity of edges routed through $e'$ in $T$.

The dependence on the capacity ratio arises from the fact that there may be many different scales of congestion on the edges of $T$. The solution is simply to replace $l$ with $l'(e) = \frac{l(e)+c(e)^{-1}}{2}$, a mixture of the original lengths with the inverse capacities. Constructing a small-stretch tree with respect to $l'$ still satisfies equation (8) with an extra factor of two, but also implies no tree edge is congested by more than $\tilde{O}(m)$.